Contents

Section outline

1. **Introduction**
   - Light switch control
   - Non-uniqueness
   - Forming Boolean functions
   - Beyond combinational logic
   - State m/c for lighting
   - Gate circuits

Light switch control

- x Boolean variable to indicate low light in room (1: low light, 0: otherwise)
- u Line to turn light in room on or off (1: turn light on, 0: turn light off)

  Let light be on if light is low: $u \leftarrow x$

  Don't want light going on and off to oscillate
- l Boolean variable to indicate light is on (1: light is on; 0: light is off)

  Let the light be on if light is low or the light is already on; $u \leftarrow x \vee l$; $u \leftarrow x + l$

  Light never goes off; would like to turn off when there's enough light
- y Boolean variable to indicate enough light outside (1: enough light outside; 0: otherwise)

  Let the light be on if light is low or the light is already on but not enough light outside: $u \leftarrow x + (l \cdot \overline{y})$; $u \leftarrow x + l\overline{y}$

Non-uniqueness

- $u \leftarrow x + l\overline{y}$
- $u \leftarrow (x + l) \cdot (x + \overline{y})$ – are these equivalent?

x	l	y	$x + l\overline{y}$	$(x + l) \cdot (x + \overline{y})$
0	0	0	0	0
0	0	1	0	0
0	1	0	1	1
0	1	1	0	0
1	0	0	1	1
1	0	1	1	1
1	1	0	1	1
1	1	1	1	1

- $u \leftarrow xx + lx + x\overline{y} + l\overline{y}$
- $u \leftarrow x + lx + x\overline{y} + l\overline{y}$
- $u \leftarrow x + x\overline{y} + l\overline{y}$
- $u \leftarrow x + l\overline{y}$
- Which one to use?

Forming Boolean functions

- We might like to have redundancy in assessing outside light
- Say, there are three sensors yielding y_1, y_2 and y_3
- How to use these?
- Go by majority: $y \leftarrow y_1 y_2 + y_2 y_3 + y_3 y_1$
- True if majority are true; false if majority are false
- $u \leftarrow x + l\overline{y} = x + l \cdot \overline{(y_1 y_2 + y_2 y_3 + y_3 y_1)}$
- $u \leftarrow x + l \cdot (\overline{y_1 y_2} + \overline{y_2 y_3} + \overline{y_3 y_1})$
- Intuitively, from the definition of majority
- By the application of De Morgan's theorem (to be studied)

Beyond combinational logic

- Suppose there is a lightning
- External lighting is high momentarily
- But we wouldn't like the light to go off – solution?
- Wait for sometime and see the external lighting stays on
- Now system works with some memory ($c = 0$: not counting, $c = 1$: counting)
- Memory is encoded in a finite number of states of the machine
- How to wait?
- Use a counter (digital) or a monoshot multivibrator (op amp based)

State m/c for lighting

A counter may be used to wait (synchronous design, using a clock)

Signals related to counter

- *z* Boolean variable to indicate all the bits are zero (1: all zero, 0: not all zero)
- *c* Line to enable count down (1: count down, 0: counting disabled)
- *r* Line to reset the counter to all 1's (1: reset, 0: normal operation)

Control states related to counter

- *N* Normal state (not counting, counter disabled)
- *S* Get ready to count (set to maximum count)
- *D* Counting down
- *C* Counting over

State m/c for lighting (contd.)

PS	Input condition	NS	Output
N	$l = 1 \wedge y = 1$	S	$u \leftarrow 1, c \leftarrow 0, r \leftarrow 1$
	$l = 0 \vee y = 0$	N	$u \leftarrow x + l\overline{y}, c \leftarrow 0, r \leftarrow 0$
S	–	D	$u \leftarrow 1, c \leftarrow 1, r \leftarrow 0$
D	z	C	$u \leftarrow 1, c \leftarrow 0, r \leftarrow 0$
	$\overline{z}$	D	$u \leftarrow 1, c \leftarrow 1, r \leftarrow 0$
C	–	N	$u \leftarrow x + l\overline{y}, r \leftarrow 0$

Mealy m/c outputs depend on the inputs and the present state

Moore m/c outputs depend only on the present state

State m/c for lighting (contd.)

A monoshot multivibrator may be used to wait (asynchronous design, not using a clock)

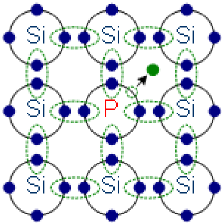
Signals related to monoshot

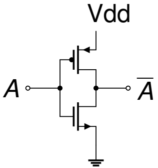
- z Boolean variable to indicate timing out (1: triggered, 0: not trigger)
- r Line to trigger the monoshot (1: trigger on, 0: trigger off)

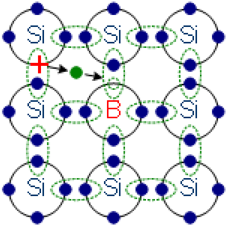
Control states related to monoshot

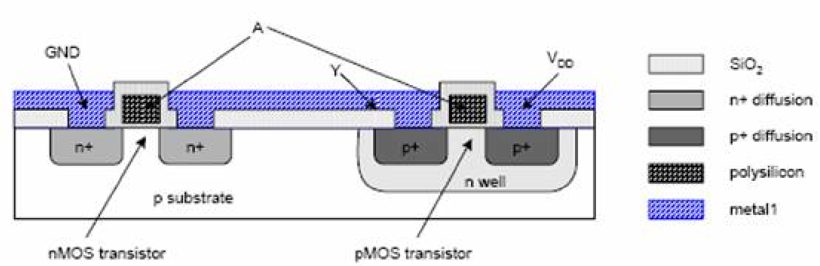
- N Normal state ($z = 0$)
- S Monoshot triggered ($r \leftarrow 1$, enter after $l = 1 \wedge y = 1$)
- D Waiting to timeout ($r \leftarrow 1$, enter after $z = 1$)
- C Timeout over (after $z = 0$); move to N

Gate circuits

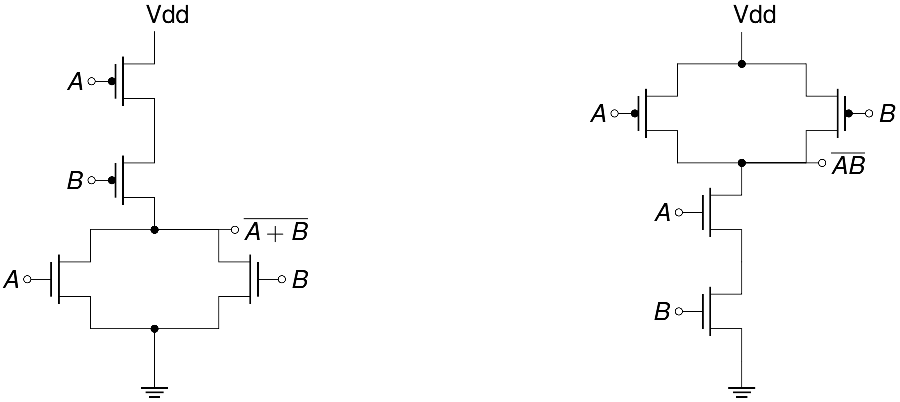




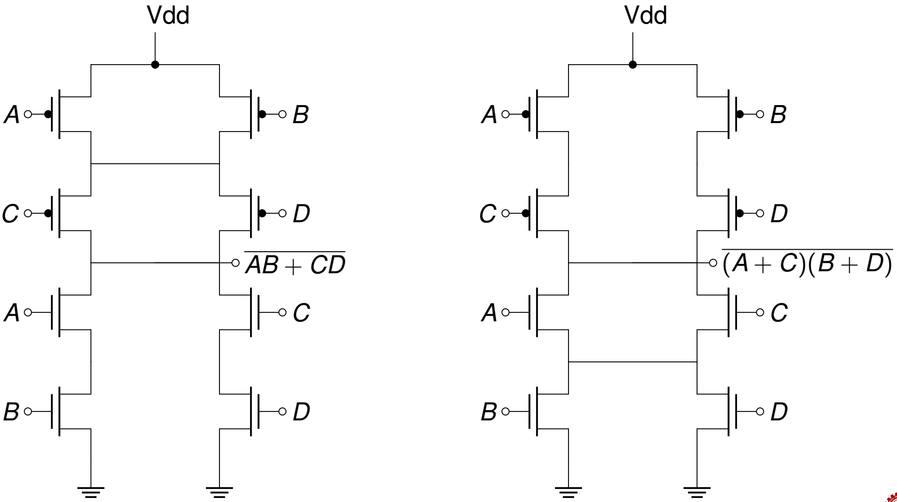




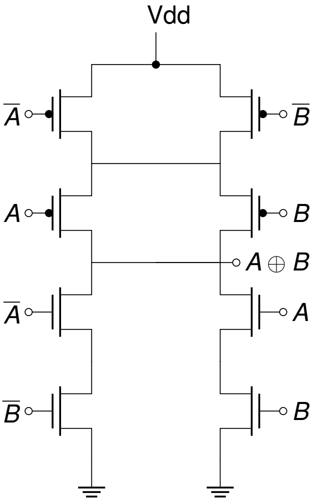
Gate circuits (contd.)

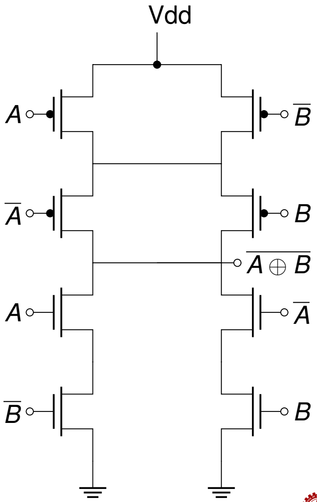


Gate circuits (contd.)



Gate circuits (contd.)





Gate circuits (contd.)